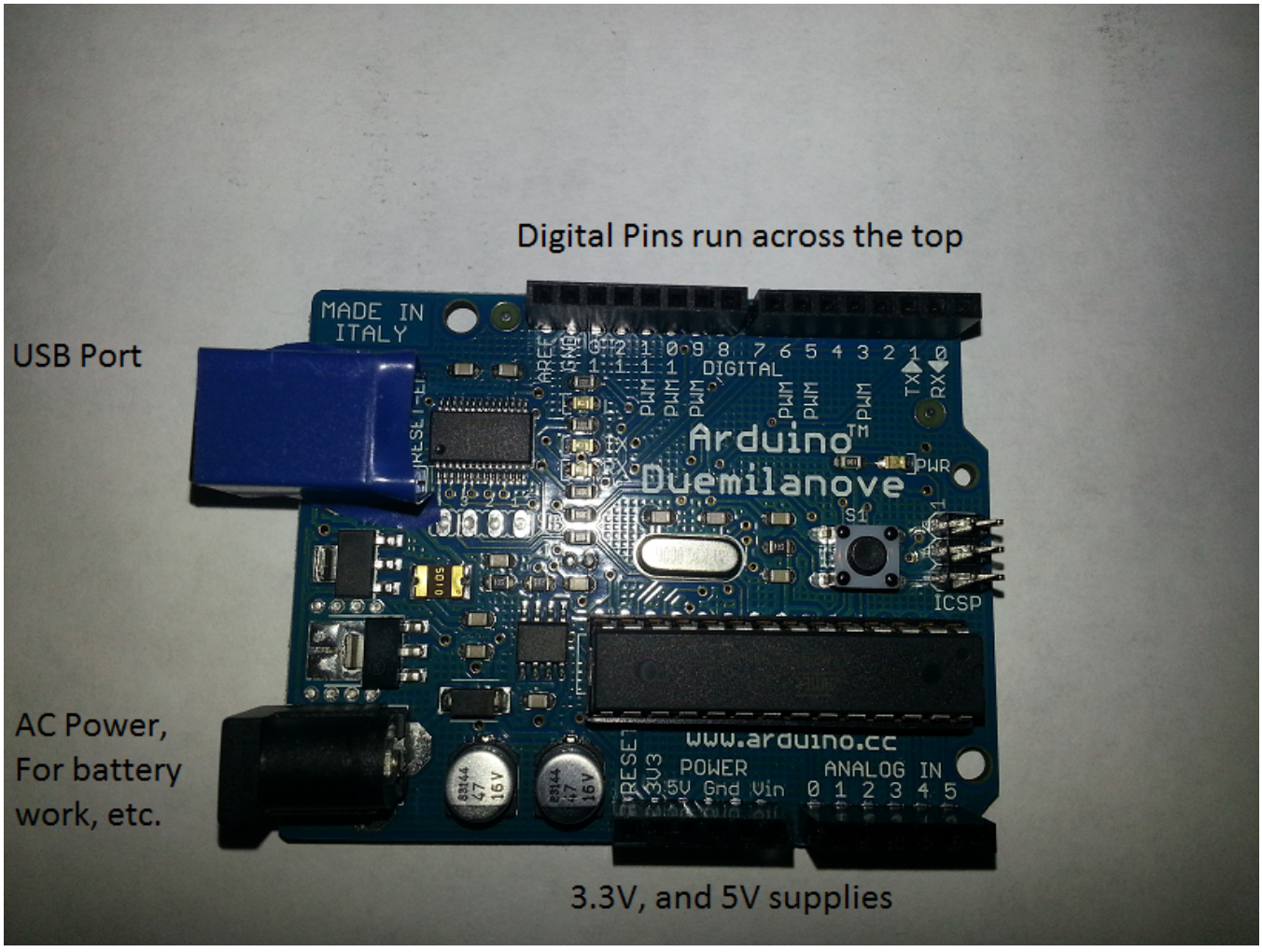
Physics 364, Fall 2012, Lab #9

(Introduction to microprocessor programming with the Arduino)

Lab for Monday, November 5

Up until this point we have been working with discrete digital components. Every NAND and OR gate had it's own chip we had to plug in ourselves. The introduction of micro-controllers which combine this functionality into a universal problem solver makes our lives much easier. In these labs we will use a micro-controller called the Atmega328(The big chip in the image below), which has been put on a prototyping platform called the Arduino. The prototyping board gives us a few handy things – a USB hookup so we can program and power our device with one cable, a built in 3.3V and 5V power supply with common ground, and presoldered connection spots. There are also expansion shields(one of which we'll use) that let you easily expand the capabilities of the Arduino.



Discrete components are ok for very simple(and perhaps fast) computation, but for

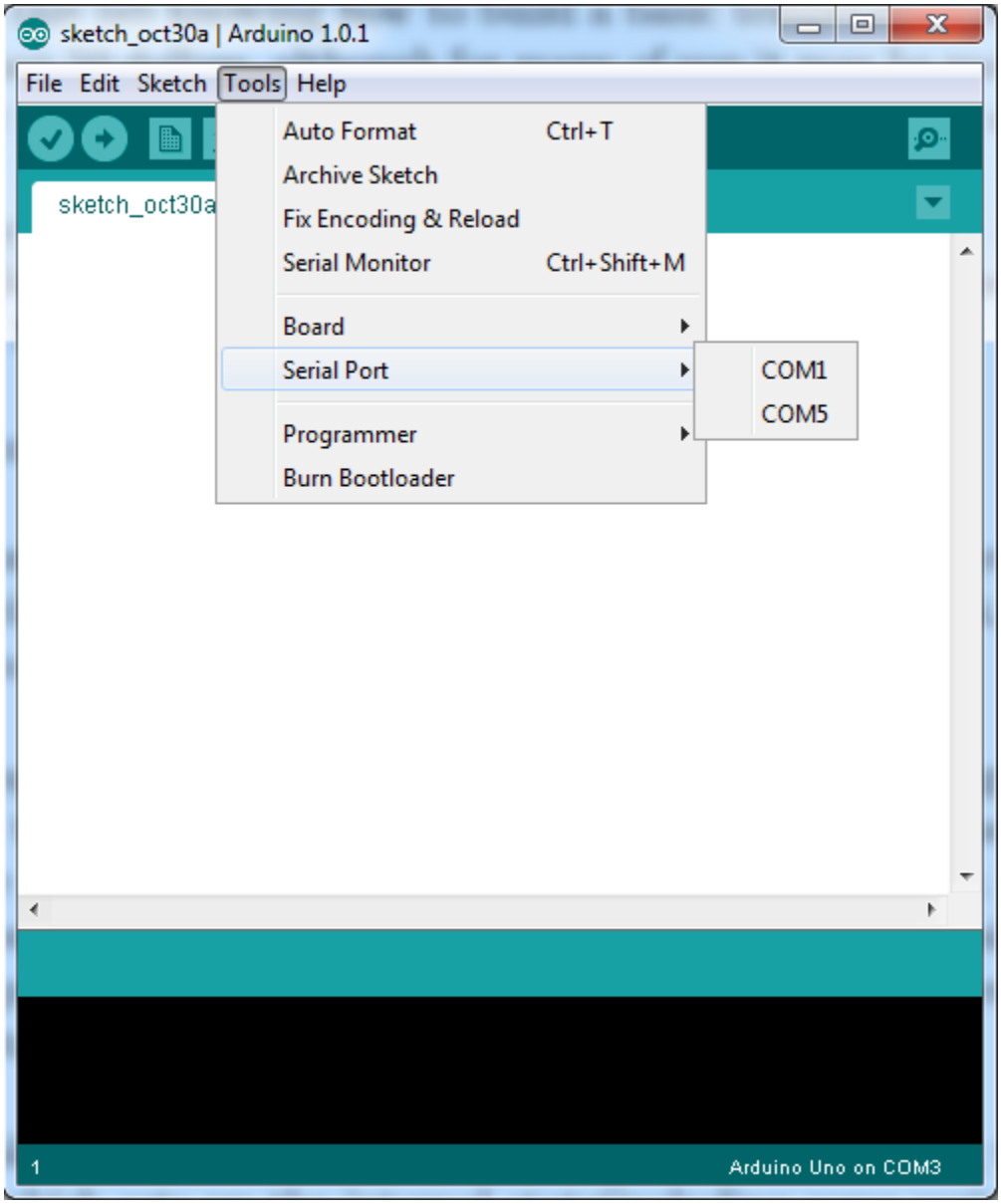
complex tasks a computer is the right way to go about it! We cannot always depend on a personal computer for these tasks though – It's wasteful (the Arduino's we work with can be yours for about 30 dollars, while a personal computer won't cost less than a few hundred), and it's often more difficult to deal with basic functionality from a PC than it is from a micro-controller (Have you ever tried to individually set the pins on your serial or parallel port outputs on your computer? It's not easy). The other advantage of a micro–controller is that it is real time. A computer will process data more or less when it gets around to it, while a micro–controller only has one job – doing what you told it to do. If your experiment needs timing accurate to anywhere near a few clock cycles, you can't use a typical computer.

The Arduino is a compromise between the desire to show you low level computation (which we will do by examining how chips talk to one another, and perhaps later in the course with FPGAs), and the desire for you to be able to DO things. We hope that you will be able to walk out of this part of the lab knowing how to build a basic temperature controller for less than 50 dollars, although for many of you it may be more important that you know how your experimental electronics work (and when they don't!).

We start this lab by getting the Arduino working on the lab computer. At home you can set it up as follows(you can follow this guide in its entirety at home, or find the equivalent ones for Mac and Linux):

`http://arduino.cc/en/Guide/Windows`

We are starting from step 5 - the arduino environment can be found in the start menu. Be sure to set the serial port and the board type to match your setup. To determine the serial port, open the program prior to plugging in your Arduino and note which ports are present. After you have plugged it in a new one will appear - this is the one you should select. To determine the board type you have, look at your board. If you don't do this correctly you will see an error when you try and upload code - a sync error being the most common. If things lock up at this stage you can usually fix them by moving the USB to a new slot and setting the board and port settings properly.



Once you have your board hooked up you should be able to compile it and then upload the example code called Blink(Under the File Menu > examples > Basics). You should see the LED next to digital pin 13 blinking at 1/2 Hz. The upload button is shown in the image below.

Every Arduino program is written in a simple Arduino specific language, and is actually converted to C behind the scenes before being converted to actual machine code. When programming a more advanced microcontroller one would typically code it in C, or even in Assembly. We thought we would spare you that grief, and get

to the fun bits right away. Besides - you can solve many simple problems using this board. It wont get you everywhere, but it will give you a strong start.

Back to our example. At the core of every Arduino program are two pieces of code: setup() sets up the internal state(including some variables and pin states) and is run prior to any other code. The second block is the loop() function. This piece of code is run continuously by the microprocessor. All logic needs to occur in this space. One function sets up the microcomputer, the other is its main program. Let's take a look at this code, this time recommented to reflect the function of each command(Any text following a // is not compiled):

```
void setup()
{
  pinMode(13, OUTPUT);
  // Setup the pin as being driven BY the Arduino,as opposed to INPUT
  // Pin 13 happens to connect to an LED on most Arduino boards.
}

void loop()
{
  digitalWrite(13, HIGH); // Sets the pin to +5V
  delay(1000); // Delay is measured in milliseconds
  digitalWrite(13, LOW); // Brings to pin back to ground
  delay(1000); // wait for another second
}
```

So this isn't too exciting, so let's spruce it up a bit. Start by saving the code as a new file, as we can't modify the existing examples. One thing we might do is to try counting. This requires we add state information to the system. We can implement a simple counter as follows(and there are many ways to do this):

```
int counter; // create a new global variable
void setup()
{
  pinMode(13, OUTPUT);
  counter = 0;
}

void loop()
{
  int delay_val =1000*(1+counter); // Variable controls the length of time we delay
  digitalWrite(13, HIGH);
```

```
  delay(delay_val); // This time delay for a variable amount of time
  digitalWrite(13, LOW);
  delay(1000);
  counter = counter + 1;
}
```

Try running this code - what happens? Note the difference between a global and a local variable: delay_val is destroyed on each iteration, while counter stores internal information across loops. We also initialize it in the setup() function.

A more desirable behaviour can be achieved by using an if() statement. These take the form:

```
If( var > 2 )
{
  do_stuff();
}
```

Or alternatively, you can perform one of two actions:

```
If( var > 2 )
{
  do_stuff();
}
else
{
  do_other_stuff(); // only done if var is 2 or less
}
```

Where do_stuff() is filled in with your own code.

Arduinos handle a pretty wide variety of boolean logic. The most important to note is ==, which is a boolean comparison(5==5 is true, 5==4 is false). This is in contrast to the = operator which is an assignment. IE, x=5; causes x to have value 5 from then on. Some other boolean operators you might use are: >, <, <=, >= and !=(not equal).

Now that you have some basic tools you should get a bit of practice.

Exercise 1: Make a program which repeats a 1 second, 2 second, 3 second, 4 second blink pattern.

Exercise 2: Make a program which performs the above cycle exactly once and then stops. Try to think of how to do this with two variables, as well as with only one.

For the Arduino labs be sure to include your code for each section in your lab writeup.

Now that you have a handle on the basics, let's try and get some more pins involved. We are going to start by representing a number in binary, and the easiest way is to use LEDs to represent each bit. So for instance if x is on and o is off: ooxo is 2 and xoxx is 11 for a 4 LED display.

Exercise 3: Wire up pins 2 to 7 to make a 6 bit LED display on a breadboard. Each pin in output mode is either High(5V) or low(0 volts) - you should check and see what yours are in practice. Remember to include resistors so you don't cook your LEDs. While limits vary between LEDs, 20 mA is a reasonable max.

Make a program which displays a slow count(say 1 per second) on your LED binary display. In order to get a specific bit of a variable, one normally uses a mask and the AND operator. For more information take a look at[1]:

`http://arduino.cc/en/Reference/BitwiseAnd`

For example, if we wanted to just read the 3rd bit we would use the command:

```
int val = x&0x8;
```

"val" now either has the value 0 or 8, but in practice we can view zero as "false" or "LOW", while anything non–zero as "true" or "HIGH", so the value stored in val could be fed directly into a digitalWrite command.

At this point we are ready to move on to more interesting things. Let's start by making use of those spare pins(notice we are avoiding using pins 0 and 1 - this is because those are used by the serial interface involved in programming the Arduino and using them can cause programming issues). Choose 2 pins to be inputs, initialize them in code, and attach them to buttons from the supply drawers. Remember to NEVER feed more than 5V into an input pin(or output for that matter) of the Arduino. This is virtually guaranteed to cook it.

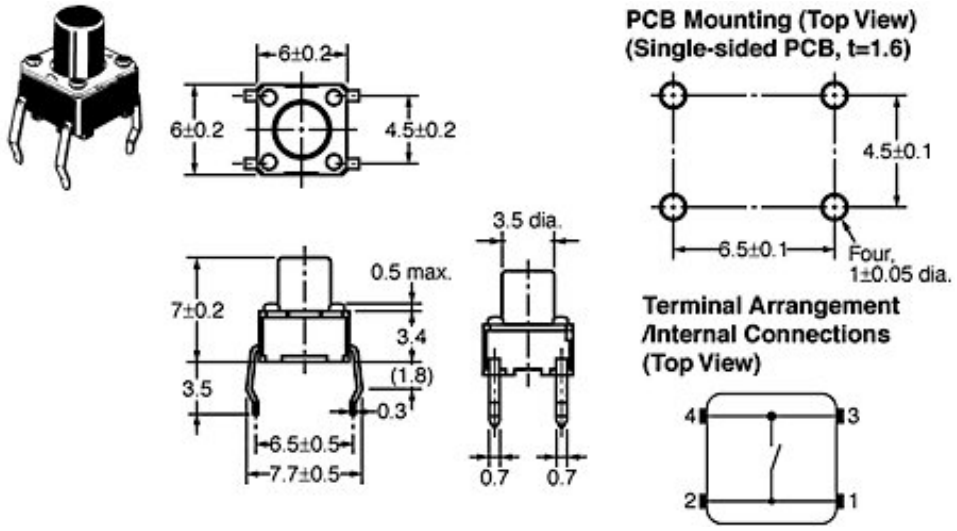
Exercise 4: Make a program which starts counting when one button is pushed and held. When a second button is pushed, reset the timer to 0. This is pretty close to

[1] Another way to do this is the bitRead command: bit = bitRead(x, n);
Where x is the variable to be read, n is the bit to be read(0 being the least significant bit) and the output is either 0 or 1 to be stored in some previously created variable 'bit'.

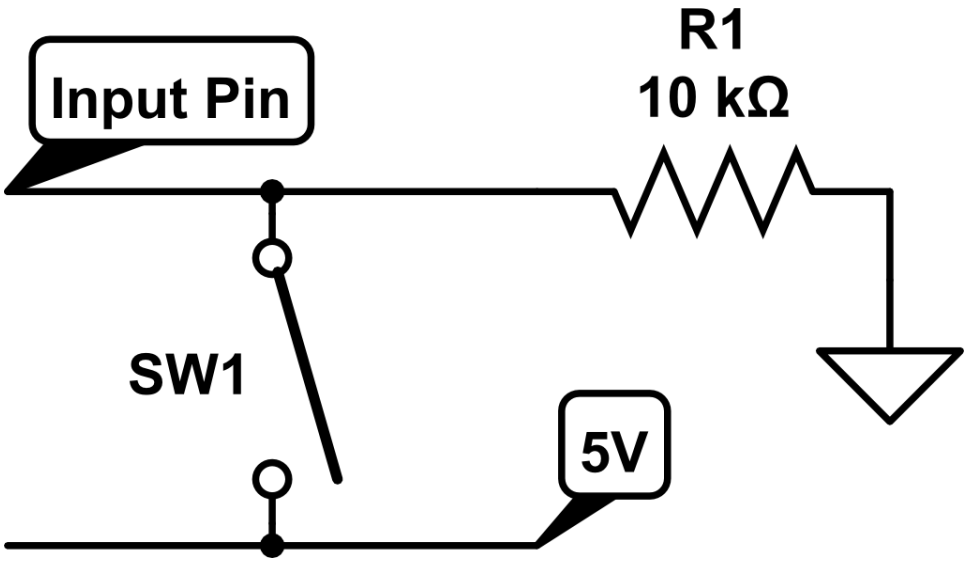
what a stopwatch does. In order to read the state of your input pins, you can use the command:

val = digitalRead(inPin);

For some variable val and for your chosen input pins. If you have time you can come back to this later and try and make this exactly like a stop watch(or at least think about how to do it). In real stop watches you push the button to make it start and then push it again later to make it stop. The same hardware can do this with a bit of extra code. The button pinout diagram is shown below:

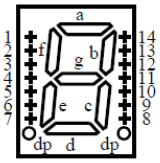


One problem you might face when using these buttons is that there is a tendency to short the 5V supply to ground through the switch. This will typically reset the Arduino, although on less forgiving electronics it might just fry the system. One way to deal with this is a pull down resistor. Rather then just connecting your switch to ground and 5V, have it connect to ground through a 10K resistor. When the switch is 'on' the drop across the resistor barely touches the voltage on the pin, while without the connection the pin is grounded through the only path available as below.



The last thing we'll do in this part of the lab is try talking to a slightly more complicated component. In practice using LEDs for counting is pretty rare. We would normally display to a computer monitor, or display the results on a 7–segment display. To get our feet wet we'll try to reproduce our counter with one of the 7-segment displays in the lab, the LN518RA. The pinout diagram can be found online, but the important part is shown below looking down from the top. The version we are using requires that we place a 5V signal to the common cathode, and then when we want to turn on a segment we bring it low. For instance, to just light the top position(labelled 'a') you need to place 5V to every pin except 1 which should be ground. You need only connect one of the common cathodes. You should confirm you can do this with direct connections before you try using the Arduino.

Terminal Connection



Pin No.	Assignment	Assignment
1	Cathode a	Anode a
2	Cathode f	Anode f
3	Common Anode	Common Cathode
4	Cathode e	Anode e
5	Common Anode	Common Cathode
6	Cathode dp1	Anode dp1
7	Common Anode	Common Cathode
8	Cathode dp2	Anode dp2
9	Cathode d	Anode d
10	Common Anode	Common Cathode
11	Cathode c	Anode c
12	Cathode g	Anode g
13	Cathode b	Anode b
14	Common Anode	Common Cathode

Exercise 5: (If you have time on Monday) As before construct a counter, only this time use the 7 segment display to count in hexadecimal to 15. In order to speed things up for you, you can find below a partially completed program with a table of values for decoding a decimal number into the 7 segment pinouts. Each value in the table

is (in order) the bit values to assign for the LEDs a through g for value 0 through D. We have intentionally left the pin assignments blank, as well as the values for E and F. If you still have time after this, you might try rewriting your code to use a for() loop(`http://arduino.cc/en/Reference/For`).

```
int counter = 0;
/* The following line creates an array of lookup values */
int lookup[16] = {  0x40, 0xF9, 0x24, 0x30, 0x19,0x12, 0x02, 0x78, 0x0, 0x18,
                0x23, 0x03, 0x27, 0x21, 0x0, 0x0 }; // fill in the zeros here

void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
}

void loop()
{
  int val = lookup[counter]; // lookup the value in the 'lookup' array

/* Insert your code for assigning bits in val to pins on your Arduino */

  delay(1000);
  counter++;
  if( counter > 15 )
  {
    counter = 0;
  }
}
```